SAP Abap + Json + MSSQL

beyhan_meyrali
Active Contributor

2023 May 11 3:57 PM

Hi Abapers,

In this blog post, I would like to share an alternative way of working with JSON, especially with huge json data.

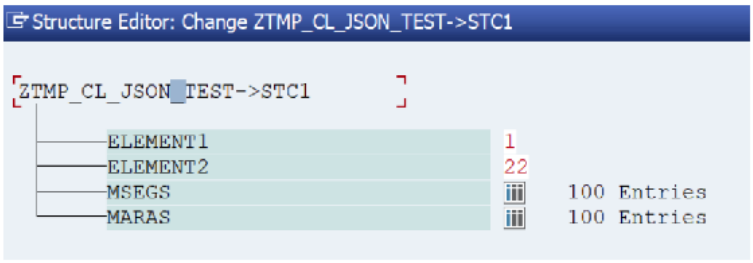
As we all experience, JSON is the most popular standard when it comes to integration, communicating to 3rd party systems. And SAP finally has good libraries to work with json(/ui2/cl_json) . But before that we need to use non-standard sap libraries for parsing, such as zJson. zJson is slower than /ui2/cl_json. Still, I thank to zJson developers for providing an option.

/ui2/cl_json is good to transform between DDIC and Json. But what you would do if you need to query json as you would query a database table? This blog post will be about that. We will see how to query a json document in MSSQL and how to use MSSQL's this ability in abap.

My scenario is, CAD software provides mbom data in json format and that is a huge file. PO converts that Json document to xml, parses that document and Sap consumes that data parts with inbound proxy calls. SAP Abap code gets data from inbound proxy and stores them in tables. Afterwards whole different data sets are processed. But Abap layer does not see original json document. That document is not stored in a database, in case if you want to compare differences between documents. And I thought, It would be great if we could store that json in a database and query it quickly directly from Abap :).

And I found out, SAP Hana provides Json Document Store and Oracle, MSSQL also provides ability to store and query json documents too. I could not try SAP Hana option, because it requires basis support. Therefore I have installed MSSQL server developer edition next to my SAP Abap Netweaver Developer edition and tested both.

First let me show you the document structure of mine and mssql database table.




```
{
    "ELEMENT1": "1",
    "ELEMENT2": "22",
    "MSEGS": [{...},{...}....n],
    "MARAS": [{...},{...}....n]
}
```

DDIC to Json

Code Block to convert DDIC to Json. I read top 100 Mara records and top 100 Mseg records and store them in a structure with CREATE_JSON_FROM_STC method.

```
* <SIGNATURE>---------------------------------------------------------
* | Instance Public Method ZTMP_CL_JSON_TEST->CREATE_JSON_FROM_STC
* +---------------------------------------------------------------------
* | [<-()] DURATION                       TYPE        DECFLOAT34
* +---------------------------------------------------------------------
  METHOD CREATE_JSON_FROM_STC.
    STC1-ELEMENT1 = '1'.
    STC1-ELEMENT2 = '22'.

    SELECT * FROM MARA INTO TABLE @DATA(MARA_TMP) UP TO 100 rows.
    SELECT * FROM MSEG INTO TABLE @DATA(MSEG_TMP) UP TO 100 rows.

    APPEND LINES OF MARA_TMP TO STC1-MARAS.
    APPEND LINES OF MSEG_TMP TO STC1-MSEGS.

    GET RUN TIME FIELD DATA(STARTTIME).

    JSON_STR = /UI2/CL_JSON=>SERIALIZE( EXPORTING DATA = STC1 ).

    GET RUN TIME FIELD DATA(ENDTIME).
    DURATION = CONV DECFLOAT34( ENDTIME - STARTTIME ).
  ENDMETHOD.

  METHOD JSON_TO_STC.
    GET RUN TIME FIELD DATA(STARTTIME).

    /UI2/CL_JSON=>DESERIALIZE(
      EXPORTING
        JSON             = JSON_STR
      CHANGING
        DATA             = STC2
    ).

    GET RUN TIME FIELD DATA(ENDTIME).
    DURATION = CONV DECFLOAT34( ENDTIME - STARTTIME ).

  ENDMETHOD.
ENDCLASS.
```

And here is the table code and content on MSSQL server. I am using PLM database and CAS table.

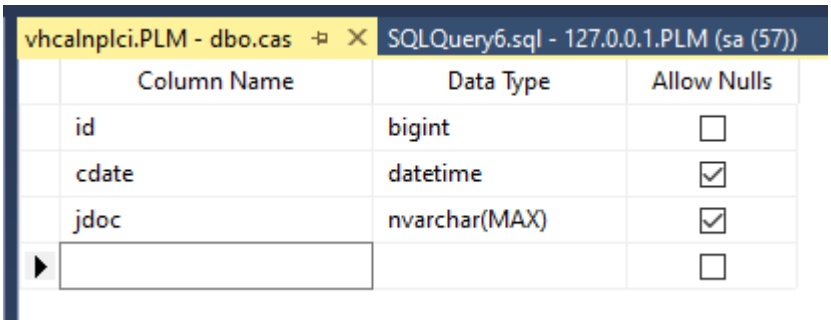


Table DDL

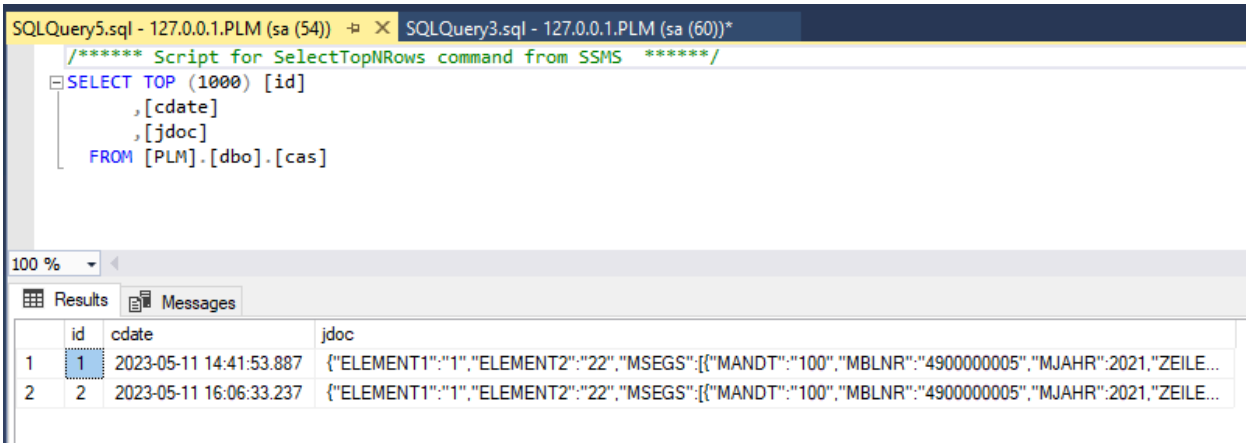
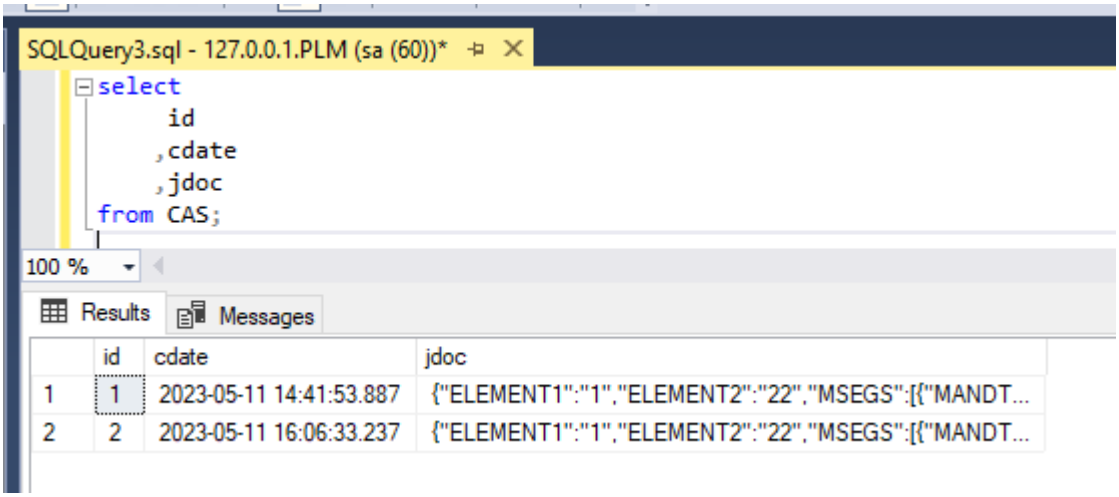
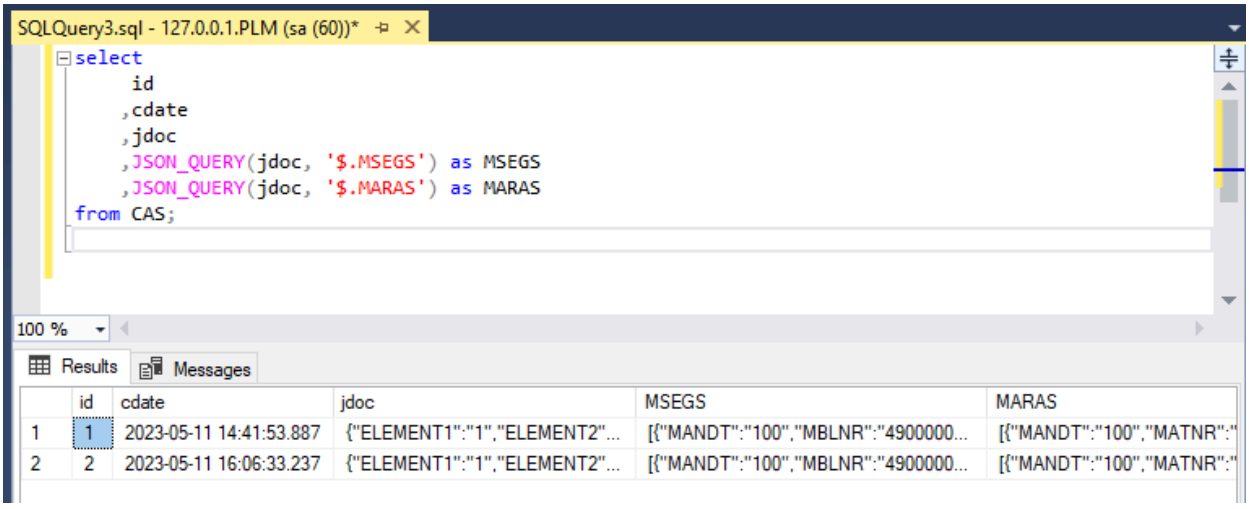
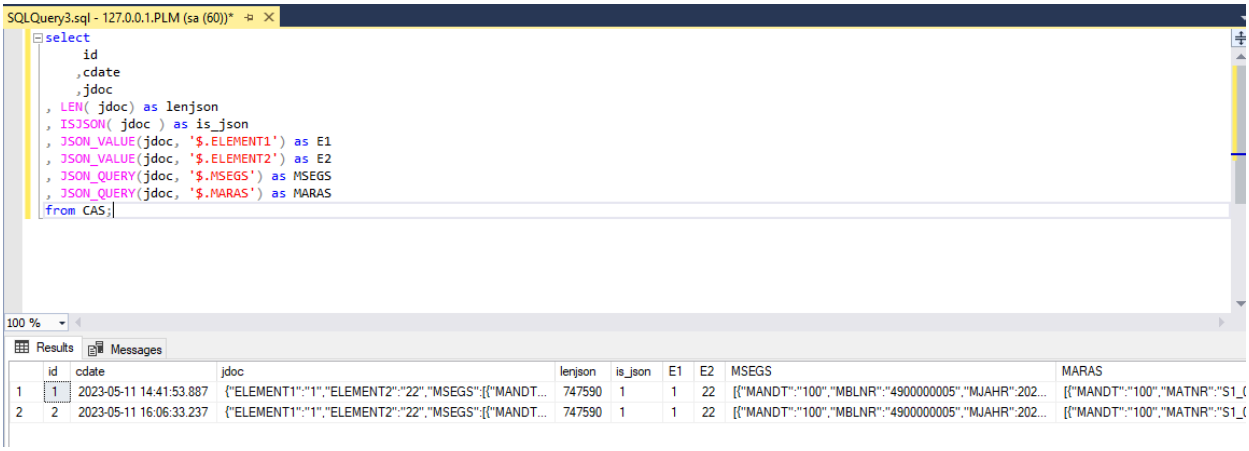


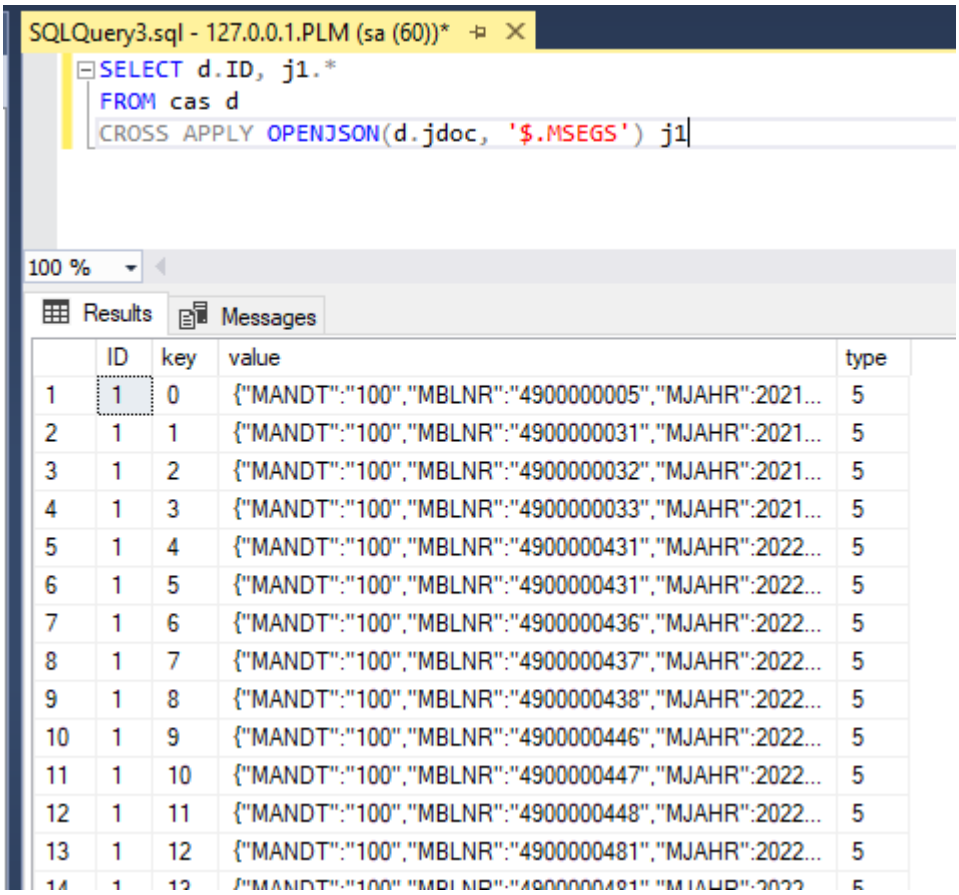
Table Content

As you can see, cas table jdoc column contains my json document. Now lets connect run a few queries and see the results.


SQLQuery3.sql - 127.0.0.1.PLM (sa (60))*
select
id
,cdate
,jdoc
from CAS;
100 %
Results
Messages
id
cdate
jdoc
1
1
2023-05-11 14:41:53.887
{"ELEMENT1":"1","ELEMENT2":"22","MSEGS":[{"MANDT...
2
2
2023-05-11 16:06:33.237
{"ELEMENT1":"1","ELEMENT2":"22","MSEGS":[{"MANDT...


SQLQuery3.sql - 127.0.0.1.PLM (sa (60))*
select
id
,cdate
,jdoc
,JSON_QUERY(jdoc, '$.MSEGS') as MSEGS
,JSON_QUERY(jdoc, '$.MARAS') as MARAS
from CAS;
100 %
Results
Messages
id
cdate
jdoc
MSEGS
MARAS
1
1
2023-05-11 14:41:53.887
{"ELEMENT1":"1","ELEMENT2"...
[{"MANDT":"100","MBLNR":"4900000...
[{"MANDT":"100","MATNR":"
2
2
2023-05-11 16:06:33.237
{"ELEMENT1":"1","ELEMENT2"...
[{"MANDT":"100","MBLNR":"4900000...
[{"MANDT":"100","MATNR":"


SQLQuery3.sql - 127.0.0.1.PLM (sa (60))*
select
id
,cdate
,jdoc
, LEN(jdoc) as lenjson
, ISJSON(jdoc) as is_json
, JSON_VALUE(jdoc, '$.ELEMENT1') as E1
, JSON_VALUE(jdoc, '$.ELEMENT2') as E2
, JSON_QUERY(jdoc, '$.MSEGS') as MSEGS
, JSON_QUERY(jdoc, '$.MARAS') as MARAS
from CAS;
100 %
Results
Messages
id
cdate
jdoc
lenjson
is_json
E1
E2
MSEGS
MARAS
1
1
2023-05-11 14:41:53.887
{"ELEMENT1":"1","ELEMENT2":"22","MSEGS":[{"MANDT...
747590
1
1
22
[{"MANDT":"100","MBLNR":"4900000005","MJAHR":202...
[{"MANDT":"100","MATNR":"S1_(
2
2
2023-05-11 16:06:33.237
{"ELEMENT1":"1","ELEMENT2":"22","MSEGS":[{"MANDT...
747590
1
1
22
[{"MANDT":"100","MBLNR":"4900000005","MJAHR":202...
[{"MANDT":"100","MATNR":"S1_(



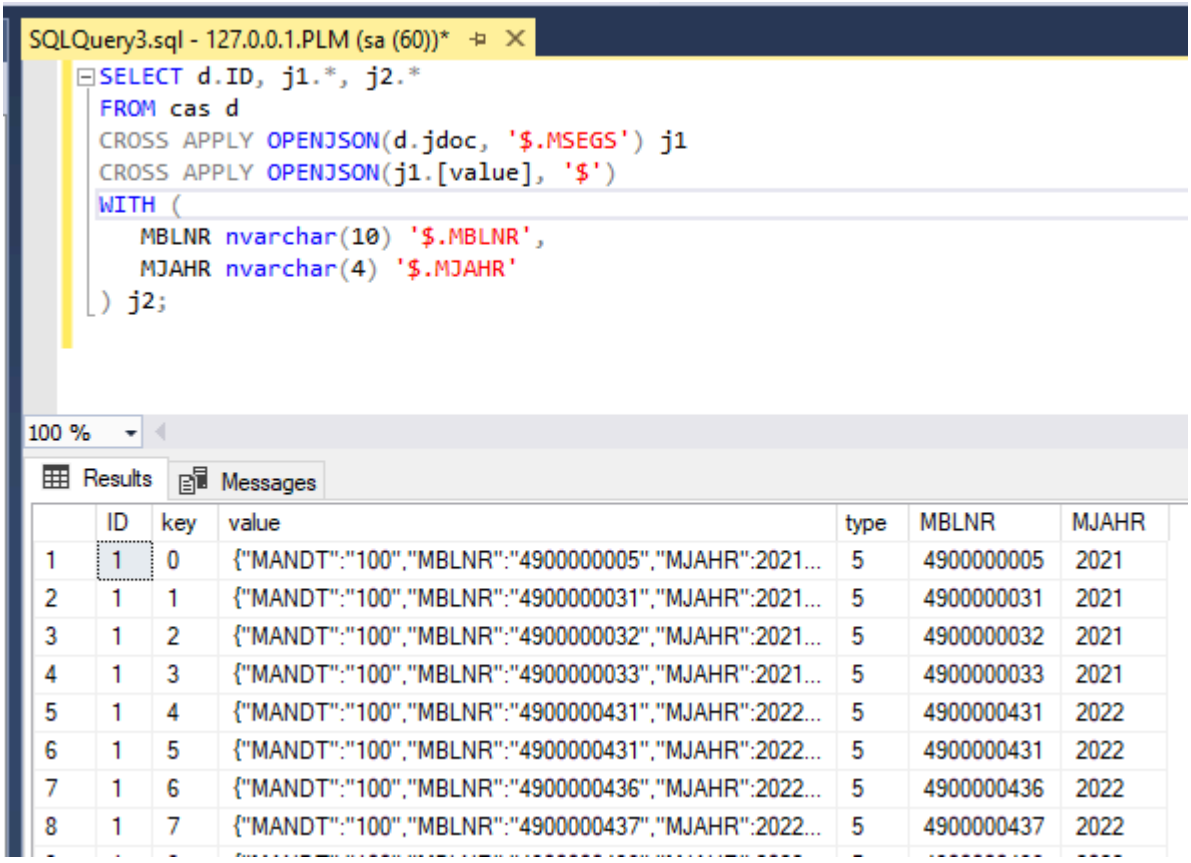
SQLQuery3.sql - 127.0.0.1.PLM (sa (60))*

```
SELECT d.ID, j1.*
FROM cas d
CROSS APPLY OPENJSON(d.jdoc, '$.MSEGS') j1
```

100 %

Results Messages

	ID	key	value	type
1	1	0	{"MANDT":"100","MBLNR":"4900000005","MJAHR":2021...	5
2	1	1	{"MANDT":"100","MBLNR":"4900000031","MJAHR":2021...	5
3	1	2	{"MANDT":"100","MBLNR":"4900000032","MJAHR":2021...	5
4	1	3	{"MANDT":"100","MBLNR":"4900000033","MJAHR":2021...	5
5	1	4	{"MANDT":"100","MBLNR":"4900000431","MJAHR":2022...	5
6	1	5	{"MANDT":"100","MBLNR":"4900000431","MJAHR":2022...	5
7	1	6	{"MANDT":"100","MBLNR":"4900000436","MJAHR":2022...	5
8	1	7	{"MANDT":"100","MBLNR":"4900000437","MJAHR":2022...	5
9	1	8	{"MANDT":"100","MBLNR":"4900000438","MJAHR":2022...	5
10	1	9	{"MANDT":"100","MBLNR":"4900000446","MJAHR":2022...	5
11	1	10	{"MANDT":"100","MBLNR":"4900000447","MJAHR":2022...	5
12	1	11	{"MANDT":"100","MBLNR":"4900000448","MJAHR":2022...	5
13	1	12	{"MANDT":"100","MBLNR":"4900000481","MJAHR":2022...	5



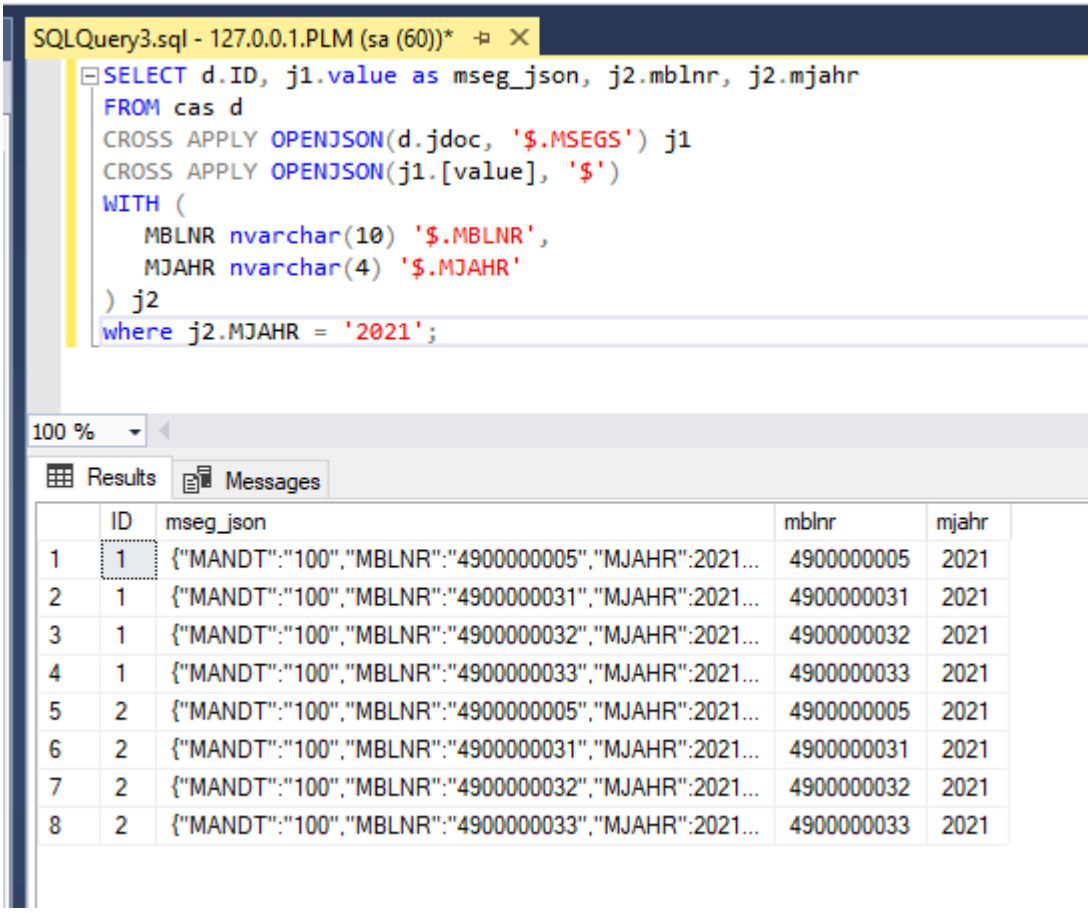
SQLQuery3.sql - 127.0.0.1.PLM (sa (60))*

```
SELECT d.ID, j1.*, j2.*
FROM cas d
CROSS APPLY OPENJSON(d.jdoc, '$.MSEGS') j1
CROSS APPLY OPENJSON(j1.[value], '$')
WITH (
   MBLNR nvarchar(10) '$.MBLNR',
   MJAHR nvarchar(4) '$.MJAHR'
) j2;
```

100 %

Results Messages

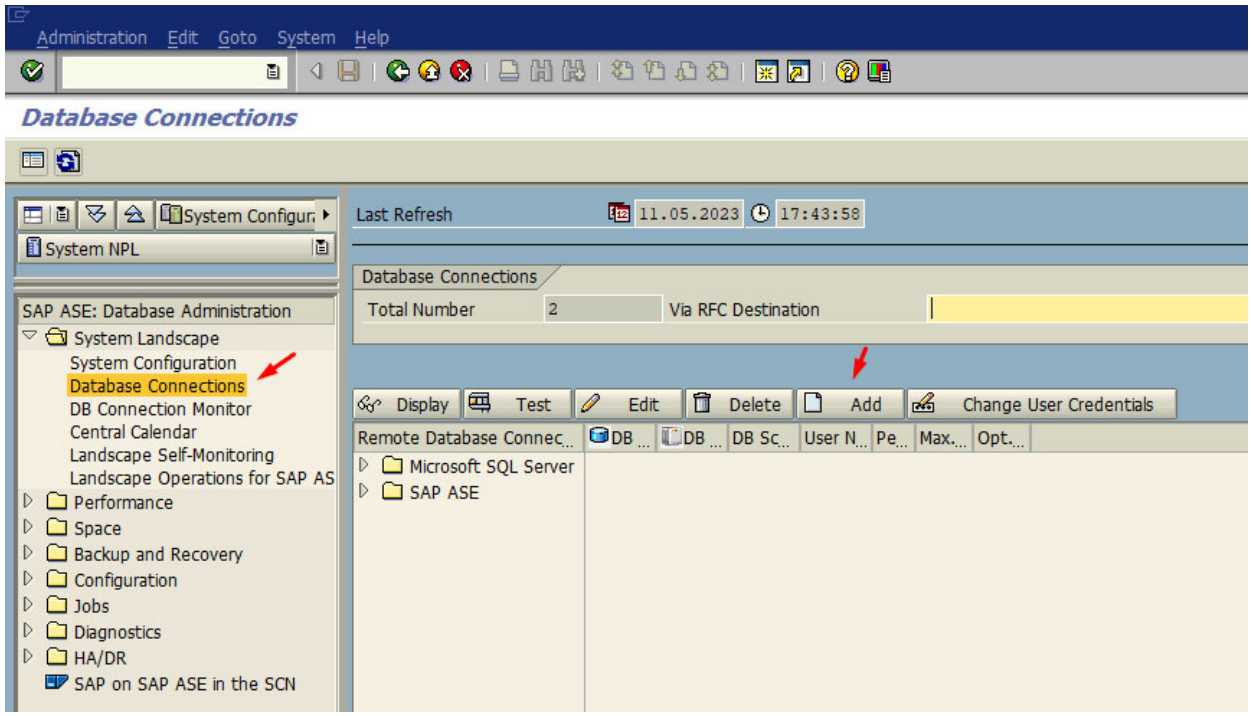
	ID	key	value	type	MBLNR	MJAHR
1	1	0	{"MANDT":"100","MBLNR":"4900000005","MJAHR":2021...	5	4900000005	2021
2	1	1	{"MANDT":"100","MBLNR":"4900000031","MJAHR":2021...	5	4900000031	2021
3	1	2	{"MANDT":"100","MBLNR":"4900000032","MJAHR":2021...	5	4900000032	2021
4	1	3	{"MANDT":"100","MBLNR":"4900000033","MJAHR":2021...	5	4900000033	2021
5	1	4	{"MANDT":"100","MBLNR":"4900000431","MJAHR":2022...	5	4900000431	2022
6	1	5	{"MANDT":"100","MBLNR":"4900000431","MJAHR":2022...	5	4900000431	2022
7	1	6	{"MANDT":"100","MBLNR":"4900000436","MJAHR":2022...	5	4900000436	2022
8	1	7	{"MANDT":"100","MBLNR":"4900000437","MJAHR":2022...	5	4900000437	2022



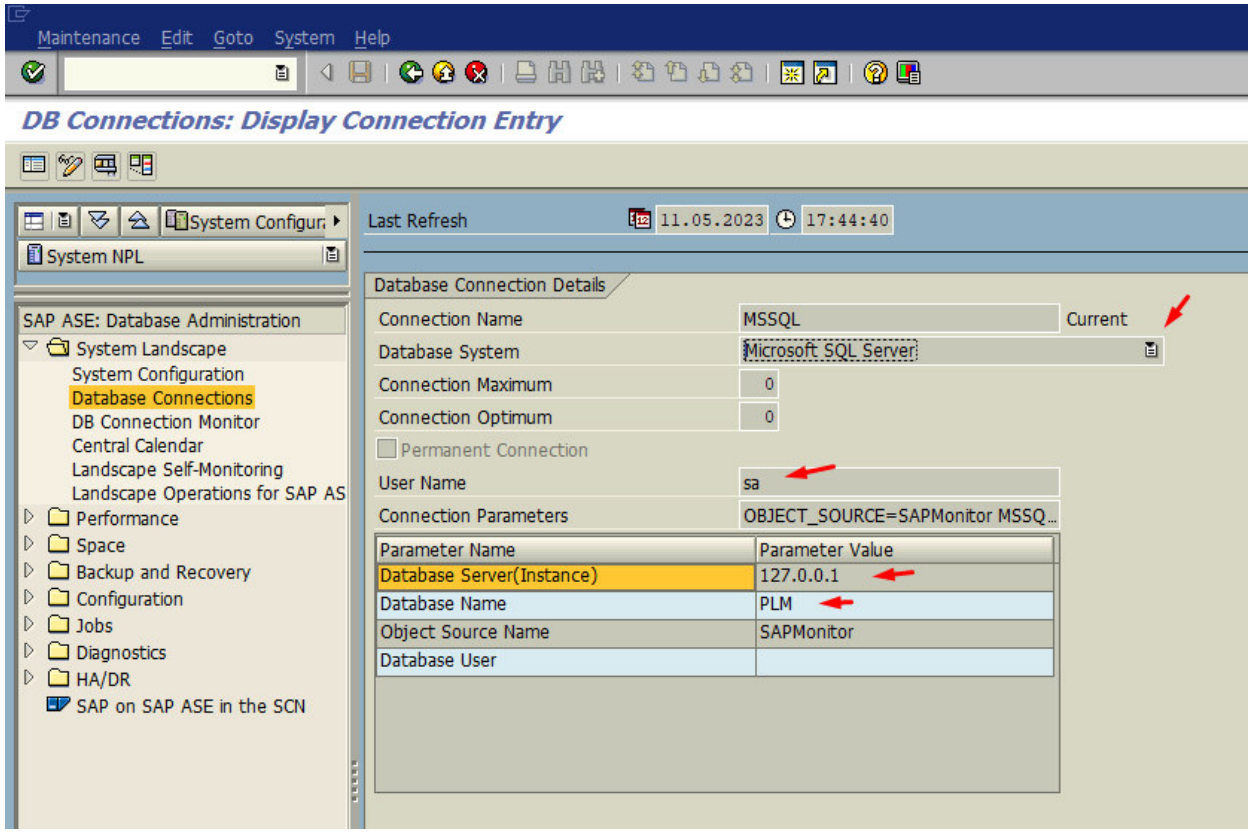
	ID	mseg_json	mblnr	mjahr
1	1	{"MANDT":"100","MBLNR":"4900000005","MJAHR":2021...	4900000005	2021
2	1	{"MANDT":"100","MBLNR":"4900000031","MJAHR":2021...	4900000031	2021
3	1	{"MANDT":"100","MBLNR":"4900000032","MJAHR":2021...	4900000032	2021
4	1	{"MANDT":"100","MBLNR":"4900000033","MJAHR":2021...	4900000033	2021
5	2	{"MANDT":"100","MBLNR":"4900000005","MJAHR":2021...	4900000005	2021
6	2	{"MANDT":"100","MBLNR":"4900000031","MJAHR":2021...	4900000031	2021
7	2	{"MANDT":"100","MBLNR":"4900000032","MJAHR":2021...	4900000032	2021
8	2	{"MANDT":"100","MBLNR":"4900000033","MJAHR":2021...	4900000033	2021

So, those are a few samples on how to query json data. You can read more on links at the Related Links section.

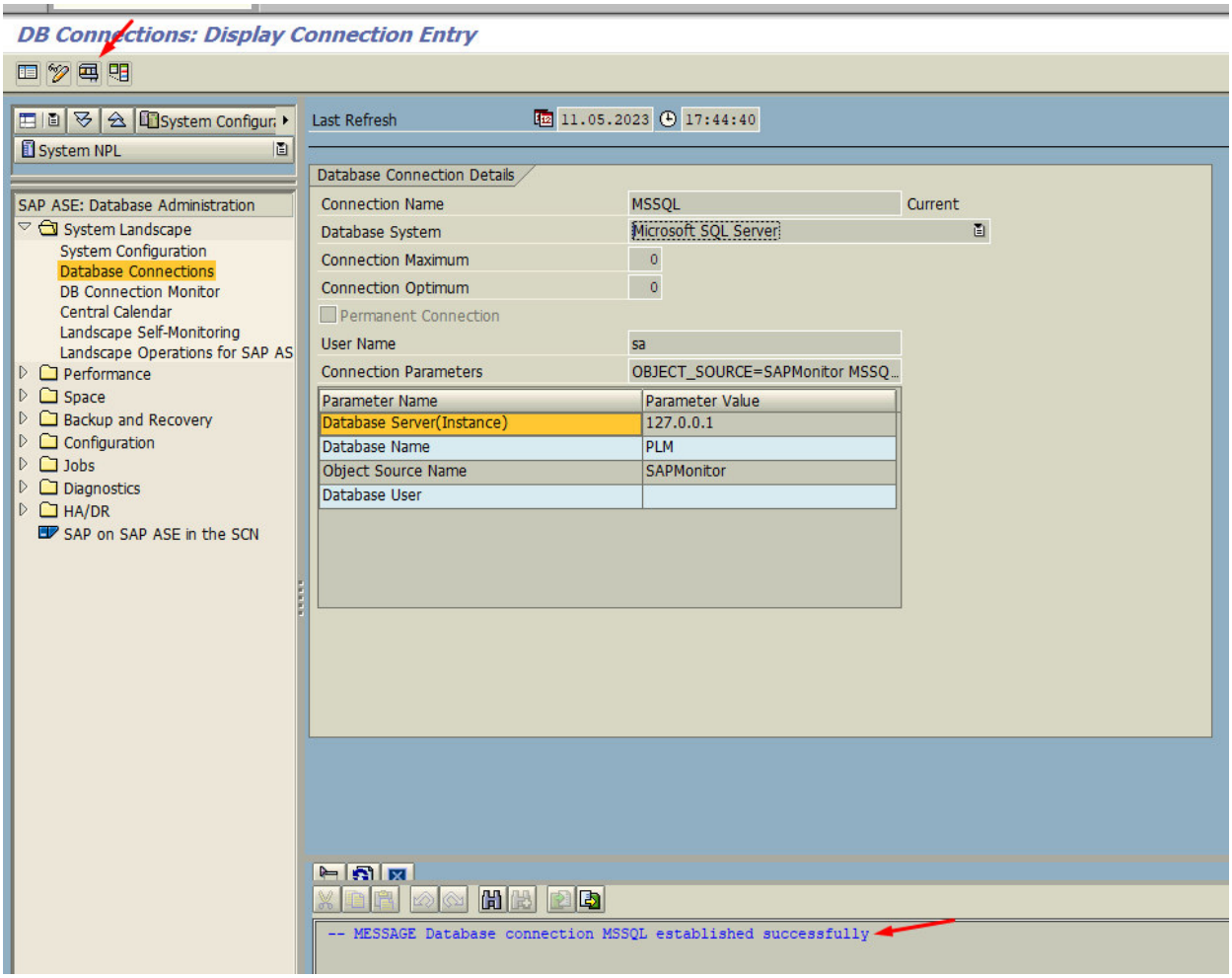
Now, lets connect SAP to MSSQL database. For that we need to open transaction DBACOCKPIT.



Create a New DB Connection



Provide Server and Database Connection Info



Test Your Connection

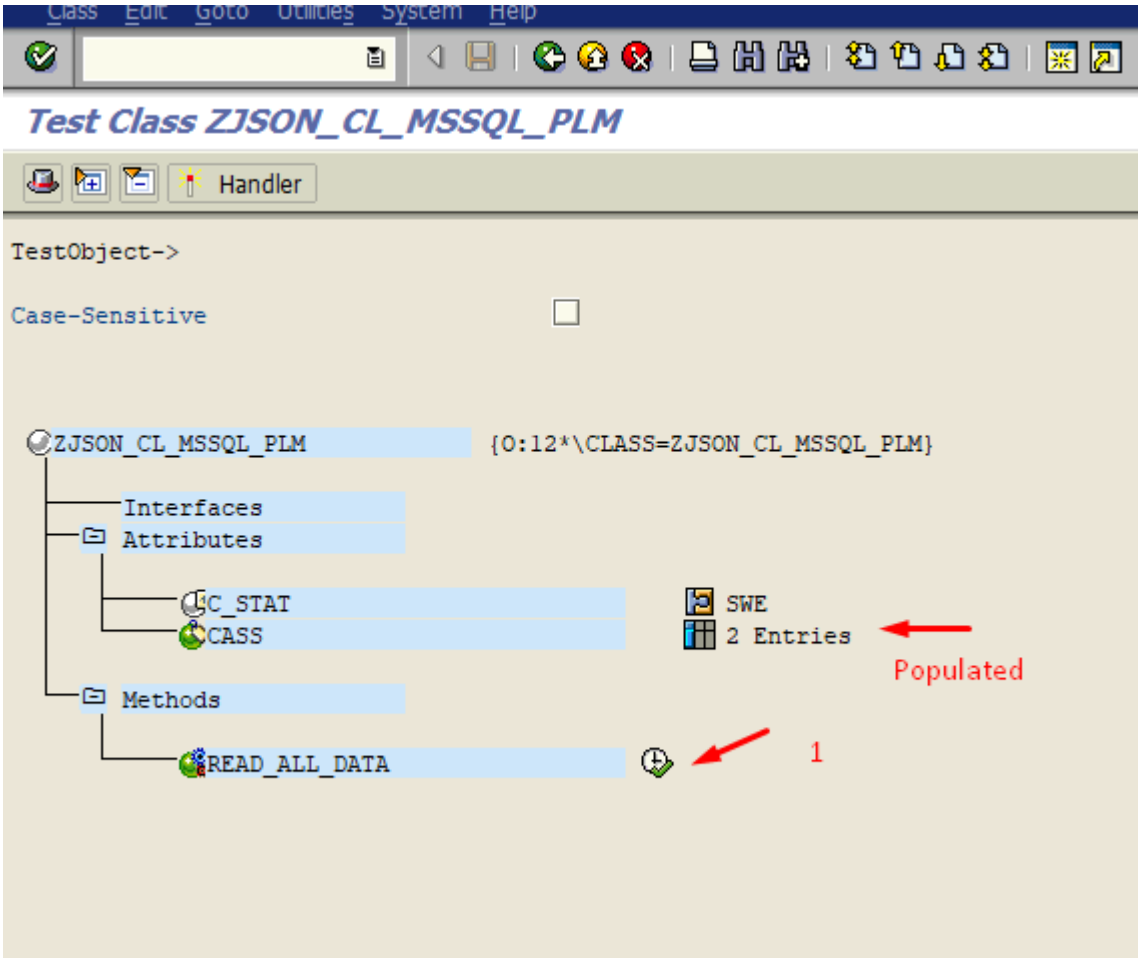
And finally a sample abap code to read directly from MSSQL with native sql.

```
CLASS zjson_cl_mssql_plm DEFINITION
  PUBLIC
  FINAL
  CREATE PUBLIC .

  PUBLIC SECTION.

    CONSTANTS:BEGIN OF c_stat,
                success TYPE char1 VALUE 'S',
                warning TYPE char1 VALUE 'W',
                error   TYPE char1 VALUE 'E',
               END OF c_stat.

    TYPES: BEGIN OF gty_status,
             status      TYPE char1,
             status_text TYPE char200,
           END OF gty_status.

    TYPES: BEGIN OF gty_cas,
             id    TYPE char100,
             cdate TYPE char100,
             jdoc  TYPE string,
             msegs TYPE string,
             maras TYPE string,
           END OF gty_cas,
           gty_t_cas TYPE TABLE OF gty_cas.

    DATA: cass  TYPE gty_t_cas.

    METHODS read_all_data
      RETURNING VALUE(status) TYPE gty_status.

  PROTECTED SECTION.
  PRIVATE SECTION.

    CONSTANTS: c_conname TYPE dbcon-con_name VALUE 'MSSQL'.
ENDCLASS.



CLASS ZJSON_CL_MSSQL_PLM IMPLEMENTATION.
```

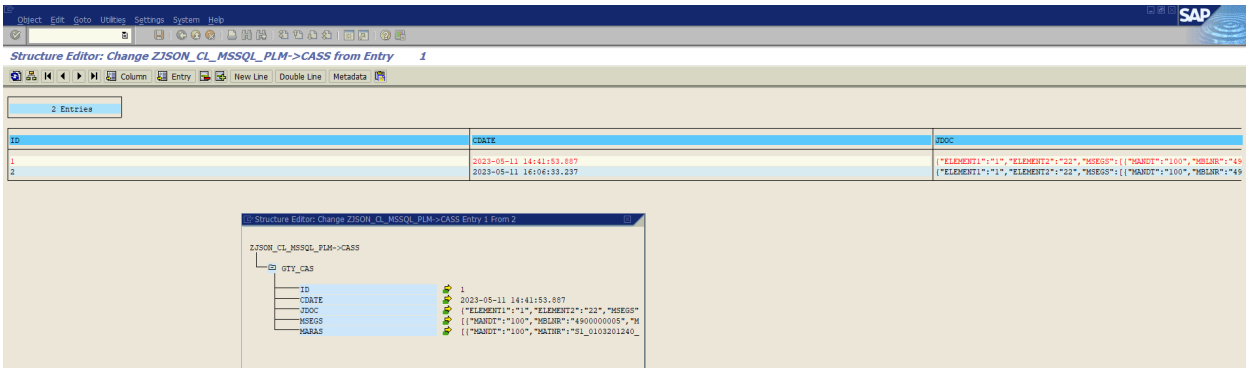
```
* <SIGNATURE>---------------------------------------------------------
* | Instance Public Method ZJSON_CL_MSSQL_PLM->READ_ALL_DATA
* +---------------------------------------------------------------------
* | [<-()] STATUS                         TYPE        GTY_STATUS
* +---------------------------------------------------------------------
  METHOD read_all_data.
    DATA: exref    TYPE REF TO cx_root.
    TRY .

        DATA: id    TYPE char100,
              cdate TYPE char100,
              msegs TYPE string,
              maras TYPE string,
              jdoc  TYPE string.

        EXEC SQL.
          CONNECT to :c_conname
        ENDEXEC.

        "Get Data
        EXEC SQL.
          OPEN dbcur FOR
            select
               id
              ,cdate
              ,jdoc
              ,JSON_QUERY(jdoc, '$.MSEGS') as MSEGS
              ,JSON_QUERY(jdoc, '$.MARAS') as MARAS
            from CAS
        ENDEXEC.

        "Fetch data
        DO.

          CLEAR:id ,cdate ,msegs, maras, jdoc.

          EXEC SQL.
            fetch next dbcur into :id, :cdate ,:jdoc ,:msegs ,:maras
          ENDEXEC.

          IF sy-subrc <> 0.
            EXIT.
```

```
            ENDIF.

            APPEND VALUE #( id = id cdate = cdate jdoc = jdoc msegs = m
          ENDDO.

          "Close db cursor
          EXEC SQL.
            close dbcur
          ENDEXEC.

          EXEC SQL.
            disconnect  :c_conname
          ENDEXEC.

          status-status = c_stat-success.

        CATCH cx_root INTO exref.
          status-status = c_stat-error.
          status-status_text = exref->get_text( ).
      ENDTRY.
    ENDMETHOD.
ENDCLASS.
```

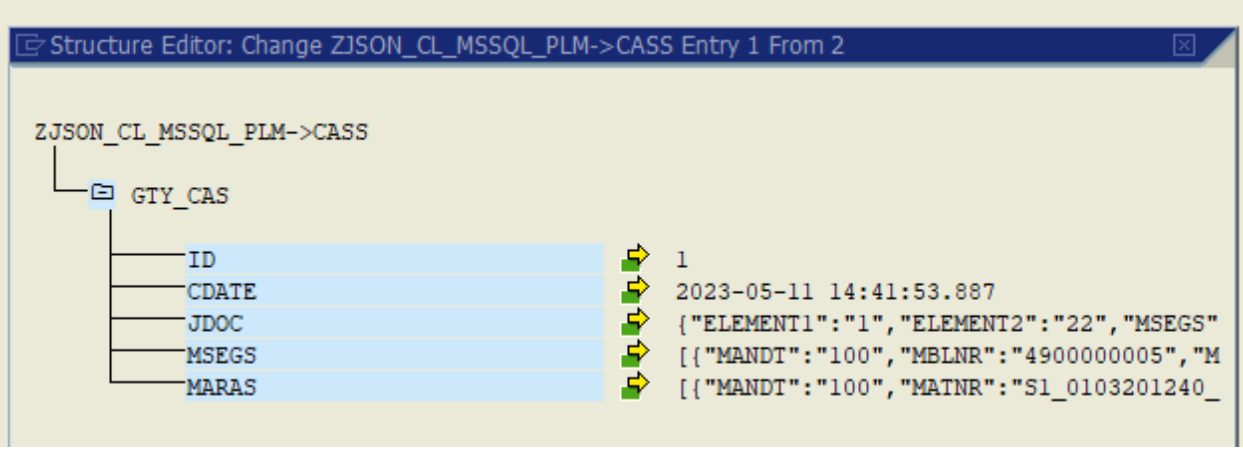
Lets run the code and see the results.



Make a Call



Results



Single Line Of Results

That is all. As you can see, we can harness power of MSSQL database and combine it with Abap. In that way instead of normalizing json data in Z tables, we can store them in database tables as json documents and query it like querying and database table.

I hope that post was helpful for you and gave you an idea.

Thanks for reading.

Regards

Related Links

https://help.sap.com/docs/SAP_HANA_PLATFORM/3e48dd3ad36e41efbdf534a89fdf278f/b4518419653e44daad99c28...

https://michals.blog/2018/10/16/performance-of-json-in-sql-server/#:~:text=JSON%20query%20took%20on%...

https://learn.microsoft.com/en-us/sql/relational-databases/json/json-data-sql-server?view=sql-server...

https://blogs.sap.com/2023/05/11/howto-migrating-from-zjson-to-ui2-cl_json/

Tags:

JSON mssqlserver

Comments

wridgeu Participant

2023 May 13
11:10 PM

Thanks for sharing this. Interesting blog post!

At the start you mentioned the speed of the JSON serialization. I think the fastest is still a custom transformation, though arguably not as easy as the simple class calls of /UI2/CL_JSON. There's a few blog posts out there (like this one https://blogs.sap.com/2022/10/27/abap-fast-json-serialization/) even comparing some more options against each other. Though not being the point you were trying to make, I thought it's nice to mention it. ^^

Cool that you included the DBACOCKPIT setup step, don't see that done often. (:

One question though for clarity: was the reason for using native SQL due to the MSSQL specific keywords that were required for the operation (JSON_QUERY)?

BR

Marco

beyhan_meyrali Active Contributor

2023 May 14
8:57 AM

Hi Marco,

Exactly.

"Open SQL allows you to access database tables declared in the ABAP Dictionary, regardless of the da...